Spring 2020

Optimizing Pollution Routing Problem

Shivika Dewan
shivika.dewan@cwu.edu

OPTIMIZING POLLUTION ROUTING PROBLEM

A Project

Presented to

The Graduate Faculty

Central Washington University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Computational Science

---

by

Shivika Dewan

June 2020

CENTRAL WASHINGTON UNIVERSITY

Graduate Studies

We hearby approve the thesis of

Shivika Dewan

Candidate for the degree of Master of Science

APPROVED FOR THE GRADUATE FACULTY

_____	_____
	Dr. Donald Davendra
_____	_____
	Dr. Razvan Andonie
_____	_____
	Dr. Szilárd Vajda
_____	_____
	Dean of Graduate Studies

ABSTRACT

OPTIMIZING POLLUTION ROUTING PROBLEM

by

Shivika Dewan

June 2020

Pollution is a major environmental issue around the world. Despite the growing use and impact of commercial vehicles, recent research has been conducted with minimizing pollution as the primary objective to be reduced. The objective of this project is to implement different optimization algorithms to solve this problem. A basic model is created using the Vehicle Routing Problem (VRP) which is further extended to the Pollution Routing Problem (PRP). The basic model is updated using a Monte Carlo Algorithm (MCA). The data set contains 180 data files with a combination of 10, 15, 20, 25, 50, 75, 100, 150, and 200 groups of cities. The optimizing techniques applied are the Discrete Differential Evolution (DDE) and, Discrete Particle Swarm Optimization (DPSO) with a `Python` Tkinter frontend. The objectives to be optimized is the fuel consumption rate and distance traveled and a statistical comparison is done between the different algorithm to compare effectiveness.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Donald Davendra, Chair of Computer Science Department at Central Washington University. The door to Prof. Davendra office was always open when I ran into problems or had a question about my research or writing. He consistently allowed this project to be my own work, but steered me in the right the direction, whenever it was needed.

I would also like to thank committee members Dr. Razvan Andonie and Dr. Szilárd Vajda, who gave great feedback during this project and guided me further after my proposal. Without their passionate participation and input, this project could not have been conducted successfully.

Finally, I must express my very profound gratitude to my parents, friends and colleagues for providing me with unfailing support and continuous encouragement throughout my years of study and the process of researching and writing this paper. This accomplishment would not have been possible without them. Thank you.

Author

Shivika Dewan

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Pollution is a big problem all over the world [1], [2]. Despite the growing use and impact of commercial vehicles, little research has been done keeping pollution as its primary objective [3]. Road transport accounts for a proportion of 92% in the United Kingdom (UK). According to [4], Freight transportation in the United Kingdom (UK) is responsible for 22% of the CO_2 emissions from the transportation sector. This amounts to 33.7 million tonnes, or 6% of the CO_2 emissions in the country. The author in [5] focuses on comparing the six models created about the green house gasses and freight transportation, and assess them with their advantages and disadvantages.

FIGURE 1: Pollution Problem in Delhi

Paper [5] also talks about the data taken from [6] which used an algorithm called Vehicle Market Model (VMM) to estimate changes to vehicle stock/kilometrage, fuel consumed and CO_2 emitted [7].

According to Demir *et. al.* [8], there are 35 organic hydro-carbon and oxygenated species that cause air pollution in the London region. Subsequently, the model created in [8], helps in lessening the affect of the loss of the ozone layer and peroxyacetyl nitrate, which could have been prevented if production of pollution was not as great.

Pollution started with the globalization of the supply chain, which resulted in an increase of transportation leading to more air pollution from carbon emissions [9]. The transportation sector itself is responsible for 24% of the overall green house gas (GHG) emissions in the EU-27 countries, of which road transportation amounts to 17%.

As shown in the Fig 1, the air quality is defined as very unhealthy. According to [10], contribution of automobiles is reported in the range of 40% to 80% of the total air pollution in Delhi, India. The anthropogenic sources of urban air pollution are classified into three major categories: industrial, mainly domestic cooking/heating and vehicular. The main cause of air pollution is fuel combustion. In India, 25% of the total energy (of which 98% comes from oil) is consumed by transport sector exclusively, which is reported to be contributing more than 50% of air pollution problem in most of the metro cities, and in some cases it was even up to 80%. As per an estimate, in 2001, air pollution contribution of transport sector was about 72% in Delhi. This makes it a major environmental issue, which is of grave concern. This research looks at one small component as to how emission can be reduced using better vehicle routing techniques between cities using evolutionary algorithms.

CHAPTER II

POLLUTION ROUTING PROBLEM

This project uses the VRP [11] as the basic model and extends this basic model to formulate the Pollution Routing Problem (PRP) [12]. The basic model of VRP is initially solved using the Monte Carlo Algorithm (MCA). The *traditional route* is the route following the order of cities given in the dataset. The MCA algorithm is used to shuffle the sequence of cities to make new routes, thus, helping to find a better route than the *traditional route*. Thereafter, the random sequence generator is applied, followed by two evolutionary optimizing techniques to ascertain if the results can be improved [13] [14]. The back-end code is written in `C++`, while `Python` is used for the front-end GUI .

Distance Calculation

Distance traveled was calculated using the formula of distance between two points in a graph using Equation 2.1.

$$\text{Distance travelled} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.1}$$

Where, *x* and *y* represents, *x* and *y* coordinates of a city. The values of *x* and *y* coordinates are given in the VRP data set. The Equation 2.1 was only valid for the VRP data set. Whereas, for the PRP data set, the distance traveled was already given in the data set in the matrix form. It will be explained further in the report Section 2.5.

Emission Calculation

The emission computation consists of a number of unique factors. For example, engine component, fuel-air mass ratio, conversion factor, engine displacement, amongst others shown in Table 1. The objective function for the emission is designated as Fuel Consumption Rate (FCR).

Fuel Consumption Rate

Pollution calculation requires the distance traveled as the core component. The equations used to create the FCR equations were taken from [12]. There were three fundermental equations:

$$FCR = \frac{\xi}{\kappa\psi}\left(kN_eV + \frac{0.5C_dA\rho v^3 + (\mu + f)v(g\sin\phi + gC_r\cos\phi)}{1000\epsilon\omega}\right) \tag{2.2}$$

Substituting, $\alpha = g\sin\phi + gCr\cos\phi$, $\beta = 0.5C_dA\rho$, $\gamma = \frac{1}{(1000\epsilon\omega)}$, $\lambda = \frac{\xi}{\kappa\psi}$ in Equation 2.2 gives Equation 2.3:

$$FCR = \lambda(KN_eV + \gamma(\beta v^3 + \alpha(\mu + f)v)) \tag{2.3}$$

$$FCR = \lambda(kN_eV\frac{d}{v} + \gamma\beta dv^2 + \gamma\alpha(\mu + f)d) \tag{2.4}$$

Equation 2.2 and Equation 2.3 is used derive to Equation 2.4, where, $kN_eV\frac{d}{v}$ is the *Engine Component* linear to travel time, $\gamma\beta dv^2$ is the *Speed Component*, and $\gamma\alpha(\mu + f)d$ is the *Weight Component* independent of speed and travel time. Some of the elements are constants and are given in Table 1.

TABLE 1: Vehicle emission parameters

Notation	Description	Value
ξ	Fuel-air mass ratio	1
κ	Gross energy of Diesel fuel (kJ/g)	44
ω	Conversion factor (g/l)	737
τ	Engine friction factor $(kJ/rev/l)$	0.2
N_e	Engine speed (rev/s)	33
δ	Engine displacement (l)	5
ρ	Air density (kg/m^3)	1.2041
A	Frontal surface area (m^2)	3.912
μ	Curb weight (kg)	6350
g	Gravitational constant (m/s^2)	9.81
θ	Road angle	0
C_d	Aerodynamic drag coefficient	0.7
C_r	Rolling resistance coefficient	0.01
ε	Vehicle drive train efficiency	0.4
ϖ	Engine efficiency parameter	0.9

Equation 2.4, is the equation that was getting used to calculate the emission on a particular route. The factor *d* represents distance traveled.

Vehicle Routing Problem

The objective of this project was to use the existing algorithms such as VRP and PRP and apply optimizing techniques such as Discrete Differential Evolution (DDE) and Particle Swarm Optimization (PSO) to see if better results can be obtained. The basic model used is the VRP [11]. It starts with the depot, whose location is (0,0) which is nothing but *x* and *y* coordinates of the depot. The tour starts from the depot and goes to different clients. If the vehicle ever visits the depot in the entire route, the current capacity changes to the max payload. When the vehicle goes from depot to the first client, the current payload changes to max payload - demand of the client visited. Then, it checks the demand of the next client and if and only if the demand is less than the current payload, the vehicle goes to the next client, else it goes back to depot and changes the current payload to full payload before going to the next client. Whenever, the vehicle

visits the depot, it is counted as a subtour. Subtour are defined as tour starting from depot and returning to the depot again. At the completion of the entire tour, the vehicle returns back to depot. The total distance is calculated by adding all the distances traveled in different subtours. Fig 2, shows an example of how the route looks like where 1, 2, 3, 4, 5, 6 and 7 are cities and depot denoted as 0 in the route. This example represents how the data set gets read and the sequence is denoted the same way as given in the data set. The final route developed was: 0, 1, 2, 3, 0, 4, 5, 6, 0, 7, 0, where 0 represents the depot and the numbers represents the cities. There are three subtours in this route, which are:

1. depot $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow$ depot.
2. depot $\rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow$ depot.
3. depot $\rightarrow 7 \rightarrow$ depot.

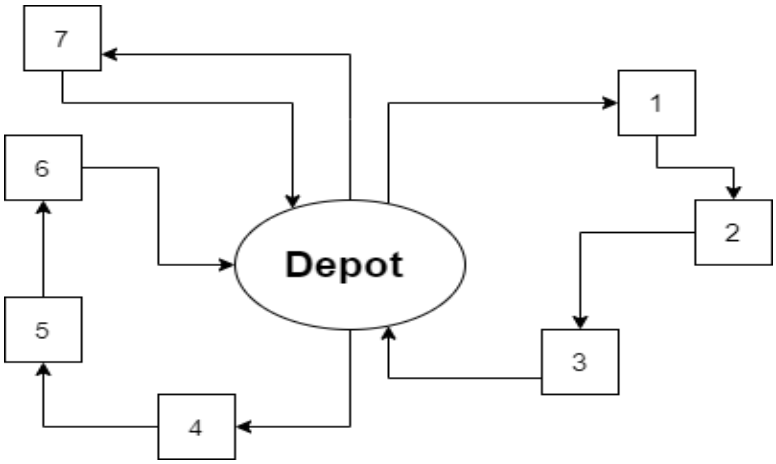

FIGURE 2: VRP

Here, the sequence represents the route in which the vehicle will travel. This is the traditional route which needs to be optimized. Fig 3, shows an outline of the VRP dataset. It has number of locations, which was 75 (given in the dataset), best result found, which

Number of locations: 75 Best: 1618.36
Capacity: 1445
0 0 : Depot Location

Locations	X Co-ordinates	Y Co-ordinates	Demands
1	35	-56	50
2	72	-58	50
3	70	-66	170
4	45	-40	297
5	39	-40	9
6	60	-50	630
7	42	-59	179

FIGURE 3: VRP Dataset

was used to compare the quality of the results. The full capacity was 1445, which is only generated when the depot is visited. The depot location is $(0, 0)$. Location are given from 1 to 75, which are the x and y coordinates with demand at each location. VRP can be better understood through Algorithm 1.

Algorithm 1 Distance traveled using VRP

```
total distance + = distance from depot to first location
current capacity = maximum capacity - demand of first location
while (Second location to end location) do
    current capacity = current capacity - demand of current location
    if (current capacity > 0) then
        total distance + = distance from current location to next location
    else
        total distance + = distance from current location to depot
        current capacity = max capacity
        total distance + = distance from depot to next location
        current capacity = current capacity - demand of current location
    end if
end while
total distance + = distance from last location to depot
Calculate FCR using Equation 2.4 for total distance
return total distance traveled
```

Pollution Routing Problem

Pollution Routing Problem (PRP) uses Equation 2.4 to calculate the CO_2 emission, which is called the Fuel Consumption Rate (FCR). The dataset used was referenced from [12]. This PRP dataset contains 180 data files with 10, 15, 20, 25, 50, 75, 100, 150 and 200 cities groups. Each city group has 20 unique datafiles. For example, UK10_01 has data of different 10 cities in comparison to UK10_02 as shown in Fig 4. The data set can be found in [15].

10-node	15-node	20-node	25-node	50-node	75-node	100-node	150-node	200-node
UK10_01	UK15_01	UK20_01	UK25_01	UK50_01	UK75_01	UK100_01	UK150_01	UK200_01
UK10_02	UK15_02	UK20_02	UK25_02	UK50_02	UK75_02	UK100_02	UK150_02	UK200_02
UK10_03	UK15_03	UK20_03	UK25_03	UK50_03	UK75_03	UK100_03	UK150_03	UK200_03
UK10_04	UK15_04	UK20_04	UK25_04	UK50_04	UK75_04	UK100_04	UK150_04	UK200_04
UK10_05	UK15_05	UK20_05	UK25_05	UK50_05	UK75_05	UK100_05	UK150_05	UK200_05
UK10_06	UK15_06	UK20_06	UK25_06	UK50_06	UK75_06	UK100_06	UK150_06	UK200_06
UK10_07	UK15_07	UK20_07	UK25_07	UK50_07	UK75_07	UK100_07	UK150_07	UK200_07
UK10_08	UK15_08	UK20_08	UK25_08	UK50_08	UK75_08	UK100_08	UK150_08	UK200_08
UK10_09	UK15_09	UK20_09	UK25_09	UK50_09	UK75_09	UK100_09	UK150_09	UK200_09
UK10_10	UK15_10	UK20_10	UK25_10	UK50_10	UK75_10	UK100_10	UK150_10	UK200_10
UK10_11	UK15_11	UK20_11	UK25_11	UK50_11	UK75_11	UK100_11	UK150_11	UK200_11
UK10_12	UK15_12	UK20_12	UK25_12	UK50_12	UK75_12	UK100_12	UK150_12	UK200_12
UK10_13	UK15_13	UK20_13	UK25_13	UK50_13	UK75_13	UK100_13	UK150_13	UK200_13
UK10_14	UK15_14	UK20_14	UK25_14	UK50_14	UK75_14	UK100_14	UK150_14	UK200_14
UK10_15	UK15_15	UK20_15	UK25_15	UK50_15	UK75_15	UK100_15	UK150_15	UK200_15
UK10_16	UK15_16	UK20_16	UK25_16	UK50_16	UK75_16	UK100_16	UK150_16	UK200_16
UK10_17	UK15_17	UK20_17	UK25_17	UK50_17	UK75_17	UK100_17	UK150_17	UK200_17
UK10_18	UK15_18	UK20_18	UK25_18	UK50_18	UK75_18	UK100_18	UK150_18	UK200_18
UK10_19	UK15_19	UK20_19	UK25_19	UK50_19	UK75_19	UK100_19	UK150_19	UK200_19
UK10_20	UK15_20	UK20_20	UK25_20	UK50_20	UK75_20	UK100_20	UK150_20	UK200_20

FIGURE 4: PRP Data Files

In Fig 5, The data-set shows the number of customers, vehicle curb weight, which is used in the FCR Equation 2.4, vehicle maximum pay load, which is used as maximum capacity, minimum speed and maximum speed, which are used as speed components in Equation 2.4. Then, distances are read in the matrix form in the code. Furthermore, node number, city names, demand for each city, ready time, due time and service time in seconds are obtained from the file. PRP works in the same way as the basic model created with VRP with random sequences. The difference between the VRP and PRP is the usage of different data sets and the calculation of FCR after each subtour is completed. Usage of

Number of Customers: 3

Vehicle Curb Weight: 6350 Vehicle Maximum Pay Load: 3650

Minimum Speed: 20 Maximum Speed: 90

Distances:

	0	1	2	3
0	0	41150	25680	54200
1	40660	0	51980	32800
2	25010	51780	0	61520
3	54270	32750	61560	0

Node Number	City Name	Demand (kg)	Ready Time (in sec)	Due Time (in sec)	Service Time (in sec)
0	Kingston_upon_Hull	0	0	32400	0
1	Pocklington	721	2171	22139	1442
2	Brough	814	644	21053	1628
3	Selby	620	1049	20424	1240

FIGURE 5: PRP Dataset

PRP is explained in Algorithm 2, where it shows how the FCR is getting computed using the standard model of calculating distance traveled.

The difference between Algorithm 1 and Algorithm 3, is that there is an addition of using subtours, which is when the vehicle visits the depot, the subtours increments by 1. The number of subtours also define the number of vehicles used. The PRP in Algorithm 2, obtains the distance traveled as a distance matrix and not computed as in the VRP as given in Algorithm 1.

Algorithm 2 PRP algorithm

```
total distance + = distance from depot to first city
current capacity = maximum capacity − demand of first city
while (second location to end location in the sequence) do
    current capacity = current capacity − demand of current city
    if (current capacity > 0) then
        total distance + = distance from current city to next city
    else
        total distance + = distance from current city to depot
        current capacity = max capacity
        total distance + = distance from depot to next city
        current capacity = current capacity - demand of current location
        subtours ++
    end if
    total distance + = distance from last city to depot
    Calculate FCR using Equation 2.4 for total distance traveled
end while
return total distance traveled and FCR
```

CHAPTER III

ALGORITHMS USED

Monte Carlo Algorithm

Monte Carlo Algorithm (MCA) used for this project utilized the Mersenne Twister (MT19937) as the pseudo-random number generator. Random sequences of cities are generated using Mersenne Twister, which is version MT19937 developed by Takuji Nishimura and Makoto Matsumoto [16]. The `C++` Mersenne Twister wrapper class was written by Jason R. Blevins on July 24, 2006 [16]. `C++` has introduced many pseudo-random number generators to replace the `rand()` function, which is used to generate random numbers, whereas one of them is the Mersenne Twister. Mersenne Twister was used for this project because it has much longer period than that of `rand()` function, which also means that random sequence will take a longer time to repeat itself [17]. Also, the statistical behavior is better as the number of iterations were high, which required the random sequence to have more randomness than standard `rand()` function. Mersenne Twister was used to generate random routes of cities in the UK. The new population was created using specific number of rows and columns. Number of columns is the number of cities read through the data file. However, the number of rows are specified by the user using the `tkinter` form. It is helpful to see if it is possible to find an optimized route using the random distribution of different cities, which could be better than the standard route used in VRP. The data set of PRP was a little different than VRP such that the VRP used to calculate distance traveled using the x and y coordinates whereas, in the PRP data set the distance traveled is already given in a matrix form. The dataset, shown in Fig 5 has been altered to 3 cities. Otherwise, the data starts with at least 10 cities. The number

of customers used for the example are three, which includes the depot as the Kingston_ upon_ hull.

Algorithm 3 MCA algorithm

```
Reading from a file the size of the Population to be created
Inputting number of columns, Number of rows from the Population
Creating a array Fitness to store the calculated f(x) value of each of row
for (from 0 to Number of Iterations) do
    Create new population using Mersenne Twister.
    calculate Distance Traveled using Algorithm 1 or Algorithm 2
    Add f(x) calculated from each row to fitness array and calculate the optimised
fitness
    int bestInd = 0;
    for (int i = 1; i < number of Rows; i++) do
        if (Fitness[i] < Fitness[bestInd]) then
            bestInd = i;
        end if
    end for
    bestFitness = Fitness[bestInd];
end for
return bestFitness
```

Algorithm 3, explains how MCA works. MCA can be used for both distance traveled and FCR as the $f(x)$ function defines which of the optimization criteria is getting used.

The two optimizing techniques used in this project are now described.

Discrete Differential Evolution

Discrete Differential Evolution (DDE) algorithm, a relatively recent algorithm, was originally developed to solve the permutation flowshop problem [13]. Besides the standard version, this algorithm is also presented as a novel discrete version. It is simple in nature, such that it takes the random sequence that gives the best results in

the population generated and tries to optimize it. The DDE algorithm consists of the following steps:

- DDE starts with initializing the initial target population, which is generated using MCA as given in Algorithm 3. $population = \pi_i = [\pi_1, \pi_2, ...\pi_{NP}]$ with size *NP*.

- To generate a mutant individual, DDE mutates vectors from the target population by adding weighted difference between two randomly selected target population member target population members to a third member at iteration *t* using Equation 3.1

$$v_{ij}^t = \pi_{aj}^{t-1} + F(\pi_{bj}^{t-1} - \pi_{cj}^{t-1}) \tag{3.1}$$

  Where *a*, *b* and *c* are three randomly chosen individuals from the target population. The only condition is that ($a \neq b \neq c \in (1, ..., NP)$) and $j = 1, ..., n$. F, where $F > 0$ is a *mutation* scale factor.

- The *crossover* operator is described in Equation 3.2.

$$u_{ij}^t = \begin{cases} v_{ij}^t & \textit{if} \quad r_{ij}^t \leq \text{CR} \quad \textit{or} \quad j = D_j \\ \pi_{ij}^{t-1} & \textit{otherwise} \end{cases} \tag{3.2}$$

  Where, the D_j is randomly chosen dimension ($j = 1, ..., n$). Each trial u_{ij}^t differs from its counterpart in the previous iteration u_{ij}^{t-1}. *CR* is defined as crossover constant in range of [0,1] and r_{ij}^t is uniform random number between $(0, 1)$.

- To decide whether u_i^t will become a member of the trial population will be decided on the bases of comparison to its counterpart target individual π_i^{t-1} at the previous generation. The selection will be based on the fitness among the trial population

and target population as given in Equation 3.3.

$$\pi_i^t \begin{cases} u_i^t \quad \textit{if} \quad f\left(u_i^t\right) \leq f\left(\pi_i^{t-1}\right) \\ \pi_i^{t-1} \textit{ otherwise} \end{cases} \tag{3.3}$$

– The above equations explains the working of DDE, however, they are only valid for data that is discrete/binary. Since, the data used for this project is neither discrete nor binary the improved equations used are given in Equations 3.4, 3.5 and 3.6.

$$V_i^t = P_m \oplus F_k\left(\pi_i^{t-1}\right) \tag{3.4}$$

$$V_i^t = P_m \oplus F_k\left(\pi_a^{t-1}\right) \tag{3.5}$$

$$V_i^t = P_m \oplus F_k\left(\pi_g^{t-1}\right) \tag{3.6}$$

Where, π_i^{t-1} is the *ith* individual from target population at iteration $t-1$; π_a^{t-1} is a randomly chosen individual from the target population at iteration $t-1$; π_g^{t-1}, which is global best solution to be saved at iteration $t-1$; P_m is the mutation probability; and F_k is the mutation operator with mutation strength of *k*.

NEH Heuristic [18] has two phases, which are explained below:

1. First phase starts with allocating job that are ordered in descending order sums of their processing times explained in Equation 3.7.

$$P_j = \sum_{k=l}^{m} p_{jk}, \qquad j = 1, ..n. \tag{3.7}$$

2. Second phase includes choosing two jobs at random, which gives two possible sequences that can be evaluated to establish the partial schedule. Then, a job

permutation is established by evaluating the partial schedules based on the initial order of the first phase.

The computational complexity of the NEH heuristic is $O(n^3m)$, which can consume a considerable CPU time especially for large data files. This can be seen further in DDE results given in Tables 11, 12 and 13.

DDE algorithm has the following steps:

1. The first step is called the *destruction phase*. This step involves removing the first sequence and calculating its distance traveled and its least fuel consumption rate and saving it for later comparison. Further, two random cities are removed from the first sequence into a partial array. The rest of the cities are kept in a separate array.

2. The second step is called *construction phase*. New sequences with new combination of both arrays are generated as shown in Fig 6 and checked to see if better results can be obtained or not.

3. The last step is called the *local search*. In this step, if the new sequence created shows better results than the old sequence, then the new sequence is added to the new generated population. Otherwise, steps 1 and 2 is repeated until a solution is found or it hits a limit where the results obtained are no longer improved. If the results still does not improve, the old sequence is saved in the new population. It can be further described as in Fig 6 and also in Algorithm 4.

These steps are repeated for the number of rows until each row is improved for the new generated population. The new population is thus generated and global best is saved.

Fig 6, is an example off how DDE works in the code. In Fig 6, the initial sequence, taken was 1, 5, 4, 2, 6, 3. Cities 1 and 4 was chosen randomly and kept in a separate array and the rest, which was 5, 2, 6, 3 was kept in a different array. Then, 1 was inserted at

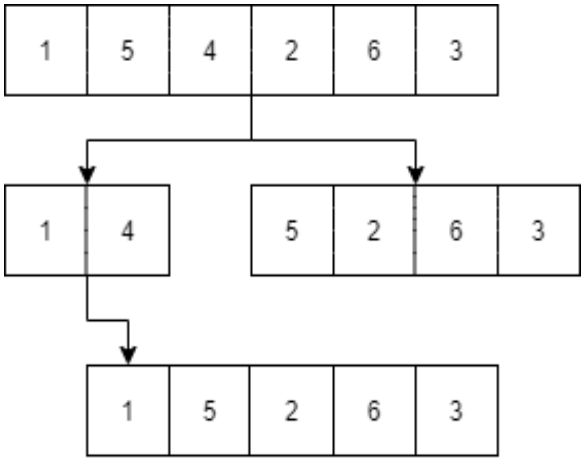

FIGURE 6: Discrete Differential Evolution

Algorithm 4 DDE algorithm

for (from 0 to Number of Iterations) **do**
  for (each sequence in the Population created from Algorithm 3) **do**
    save two random indexes from current sequence
    first array = [current sequence[index1] and current sequence[index2]]
    second array containing rest of the cities other than the first array make new combinations including first and second array
    calculate distance traveled from 2
    calculate FCR
    if (new distance traveled $<$ old distance traveled from current sequence) **then**
      old distance traveled = new distance traveled
      save new sequence created
      Update the Population generated with new sequence
    end if
  end for
  Calculate the smallest distance traveled and FCR from the updated new Population
return new smallest distance traveled and new smallest FCR
end for

position 0 and distance traveled was calculated followed with the FCR. Later, 1 was added at position 1 and the sequence was changed to 5, 1, 2, 6, 3 and so on until distance traveled and FCR stopped improving. The same procedure was applied with the other city, which is 4 in this example. New distance calculated is saved only when it improves in comparison to itself or the initial distance traveled. If the new distance never improved, then the initial sequence is the optimized sequence. Algorithm 4, uses the Population generated from Algorithm 3 and improves it until new better sequences is found for each of the given rows. The update of the sequence implies finding smaller distance traveled and a smaller FCR, which is the aim of the project. DDE does that for the given number of iterations.

Discrete Particle Swarm Optimization

The final algorithm used in this project is the Discrete Particle Swarm Optimization (DPSO) algorithm. It is a hybrid version of the traditional Particle Swarm Optimization (PSO) algorithm [14] where, real number encoded values are used. In DPSO, the population is initialized by two different heuristic procedures so that the solution quality and diversity can both be considered. That is, the first solution is obtained by the NEH heuristic [18], while the other particles can be randomly generated using the MCA as given in Algorithm 3. This project uses the random generated method to generate the particles. Particle Swarm Optimization (PSO) was generally used to solve continuous optimization problems. According to [14], when the continuous PSO was applied to discrete combinatorial optimization problems, a transformation method (e.g., the SPV rule of Tasgetiren et al. [19]) was needed to translate the continuous particle into a discrete solution. To avoid this extra work some researchers found an update to the traditional PSO and created DPSO, which proved to give better result [14]. This kept the

self-adaptive factor as one of the major factors while implementing the algorithm. There are two main steps to be followed while working on DPSO, velocity and particle update with self-adaptive perturbation. Each of them have some steps to follow, but the results are proven to be better when it comes to applying it on permutative sequences. DPSO consists of the following procedures:

1. Velocity : Following are the steps to have the velocity of each of the sequence in the original pollution. The velocity update equation is taken from [14]:

$$V_i^{t+1} = (C_1 \times V_i^t) + (C_2 \times (p_i^t - X_i^t)) + (C_3 \times (g^t - X_i^t)) \tag{3.8}$$

Where, V_i^{t+1} is the new velocity vector found,
p_i^t is personal best from each iteration t,
g^t is global best from each iteration t,
X_i^t is the current row from the population generated in which i determines the row and t determines the iteration working,
C_1, C_2, C_3 are pre-determined numbers in range $(0, 1)$.

   (a) Subtract Operator $(-)$: The subtract operator can be best explained using an example as shown in Fig 7. Given X_i^t and p_i^t, if the numbers of the sequence repeat in the same position then the subtract operator will put 0 in the same position. In the example Fig 7, at 0 the position 4 was repeated in both the sequence, so in the subtract operator the value input will be 0. If the values are not repeating, then the value from the personal best sequence will be copied to the subtract operator sequence generated.

   (b) Multiply Operator $(\times)$: Let a denote a number from the non zero elements from the subtract operator sequence. For this example, let's assign $a = 8$.

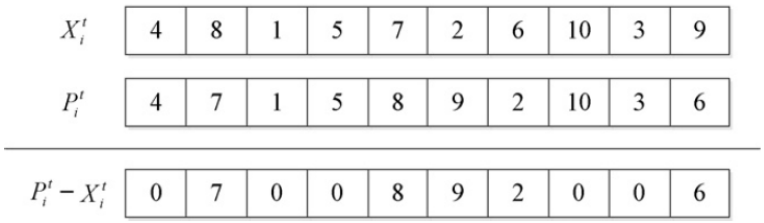

FIGURE 7: Subtract Operator for DPSO

Randomly select $\lceil aC_1 \rceil$, $\lceil aC_2 \rceil or \lceil aC_3 \rceil$ and set all the other elements to 0. It can be further explained using the example given in Fig 8 where, 8 was chosen as a and the 4th position was set to 0.

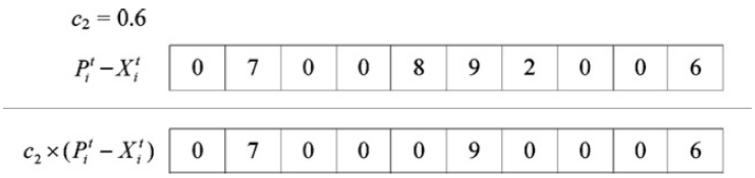

FIGURE 8: Multiplication Operator for DPSO

(c) Add Operator $(+)$: It can be explained using the example given Fig 9. There are three cases:

i. If there is only element that is non-zero, then copy the element. For example, in Fig 9, in the 0th position of all the given arrays, 4 is copied because other two elements are 0 in the 0th position of the other elements.

ii. If there is repetition of the number. For example, in the 4th position, the numbers in the three array are 2, 0 and 2. If the result array doesn't have number 2, then 2 is copied to the 4th position in the result array.

iii. If more than one element is non-zero and the element can be repeated in the result array as given in the 3rd position, then 0 is copied as the

result. Else, if more than two elements are non-zero and elements are not repeating, follow these steps:

A. Generate a number r in range $(0, 1)$.

B. if $r < C_1/(C_1 + C_2)$, then copy the first sequence element.

C. Else, copy the second sequence element. For example, in the 1st position, 3 is copied instead of 1.

iv. If more than two elements are non-zero, then follow the following steps:

A. if $r < C_1/(C_1 + C_2 + C_3)$, copy from the first sequence.

B. if $C_1/(C_1 + C_2 + C_3) <$ r $< (C_1 + C_2)/(C_1 + C_2 + C_3)$, then copy from second sequence.

C. if $r > (C_1 + C_2)/(C_1 + C_2 + C_3)$, then copy from the third sequence.

FIGURE 9: Add Operator for DPSO

2. Particle update with self-adaptive perturbation: After determining the velocity of each of the sequence, the next step is to update the old sequence using Equation 3.9.

$$X^{t+1} = (X_i^t \oplus V_t^{t+1}) \odot (C_4 \otimes R_i^{t+1}) \tag{3.9}$$

Where, C_4 is perturbation probability between $(0, 1)$,

R_i^{t+1} = (r_1, r_2) is a random insertion move, which deletes the job at position r_1 and inserts it to a different position r_2. Operators for particle update are:

(a) Update Operator ($\oplus$): There are two steps for this operator:

i. Remove the values that are V_i^{t+1} which matches X_i^t.

ii. Add the rest of the elements to the empty spaces. As shown in the example Fig 10: 8, 1, 9 is removed because these elements were getting repeated and the rest are copied into the final sequence.

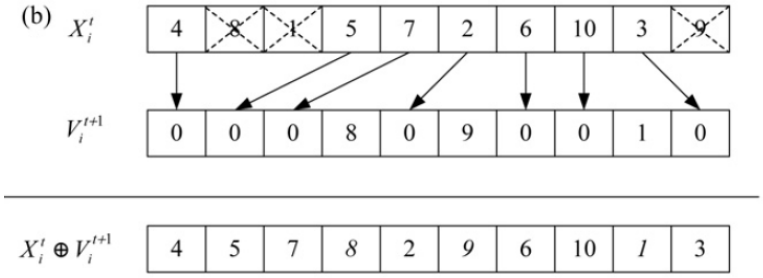

FIGURE 10: Update Operator for DPSO

(b) Self-Adaptive Perturbation ($\otimes$): Generates a random number r in range of $(0, 1)$ and if $r < C_4$, then generate a random insertion and move R_i^{t+1}. C_4 is a perturbation probability, which was obtained from [14] and is given in Algorithm 5.

(c) The operator for implementing the random insert move ($\odot$): It just applies the random insert move R_i^{t+1} on the result of $(X_i^t \oplus V_t^{t+1})$. just note that if there was non random moves that were inserted then, the operator is omitted.

The algorithm for DPSO is given in Algorithm 6.

Algorithm 5 Self-Adaptive Perturbation

$sum = 0$, $t = 0$, $rnd = 0.0$
for (int $i = 0$; $i < Rows - 1$; i++) **do**
  for (int $j = i + 1$;$j < Rows - 1$; j++) **do**
    for (int $k = 0$; $k < Columns$; k++) **do**
      $t = Population[i][k] - Population[j][k]$;
      if ($t < 0$ or $t > 0$) **then**
        sum++;
        $t = 0$;
      end if
    end for
  end for
end for
$DivCoe$ = $(float)((2 \times sum)$ / $(Rows \times (Rows - 1) \times Columns))$;
C_4 = $expf^{(-K \times DivCoe)}$; where K is a user defined input

Algorithm 6 DPSO

Generate Population using Algorithm 3
Calculate velocity of each sequence in the Population generated
set Best Personal Fitness
set Best Global Fitness
for from 0 to $iterations$ **do**
  Calculate the value of C_4
  for from 0 to $Rows$ **do**
    Update the $Velocity$ calculated
    Update the $sequences$
    Get smallest fitness
  end for
  set the Best Fitness
  set Best Personal Fitness
  set Best Global Fitness
end for
return Best Global Fitness

CHAPTER IV

EXPERIMENTS

Design

This section details the code design, which is written in the `C++` language with a `Python` frontend. In the `C++` main function, the initializing and seeding of the random generator, which is Mersenne Twister is done. Thereafter, it creates the instance of the MCA algorithm as given in Algorithm (2). It then creates the instance of Population class and creates a new population as given in Algorithm (3). The next step is to create the instance of the optimization algorithm selected, which can be either DDE or DPSO and finally the selected algorithm is run for the given number of iterations for each file. Before termination of the code, all dynamically allocated memory used is released.

Inputs for the results

The input parameters used for running the simulations were:

1. Number of solutions: 50
2. Number of columns: size of the data files (cities).
3. Number of Iterations: 100
4. The total number of times the code was executed: 180 x 3 (all three algorithms).

Hardware Details

The hardware details used to run experiments for this project is give in Table 2.

TABLE 2: Hardware Details

Operating System	Windows 10 Enterprise
CPU	Intel(R) Core(TM) i7 CPU
RAM	16.0 GB

Tkinter Forms

`Tkinter` was used as the front end for this project. It is a standard `Python` interface to the `TK GUI` toolkit [20] and is relatively easy to use. The forms created will let the user choose the *data file*, number of *rows* for the population, number of *iterations/generations* and the algorithm that the user wants to run. The `Python` code will generate a `script` file for the `C++` code and it will run the code according to the user inputs and generate results as needed. Initially, three forms were created, a *parent* form or *main* form, which then directs the user to the *input* and the *result* form.

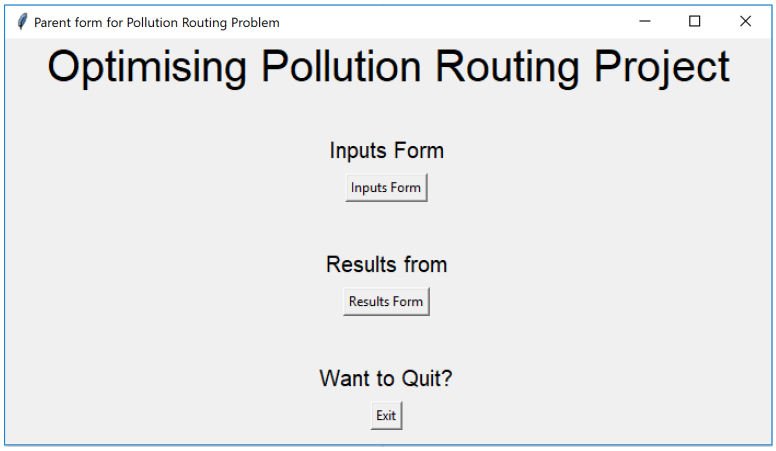

FIGURE 11: Tkinter Main form

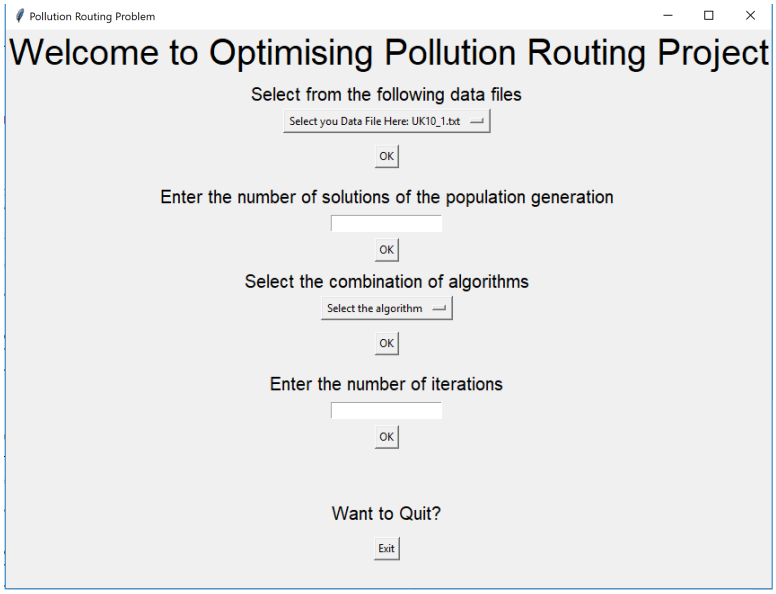

FIGURE 12: Tkinter Inputs form

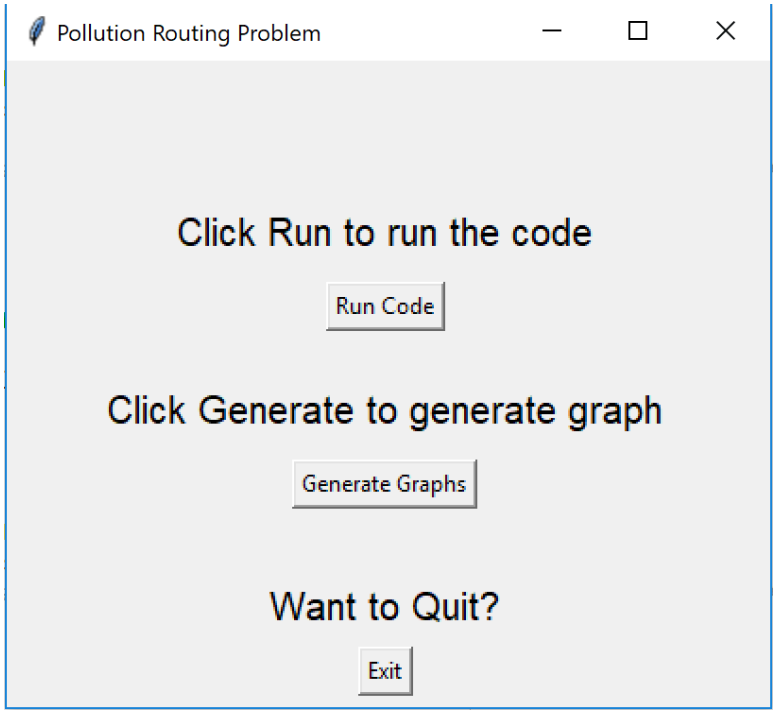

FIGURE 13: Final Tkinter Results form

CHAPTER V

RESULTS

This section gives the results obtained from the three algorithms of MCA, DDE, DPSO for the PRP dataset problems. The size of iterations was kept to 100 and number of solutions in the population was 50.

The results are given for all three algorithms and compared on the basis of their standard deviations, averages and CPU total runtime. The runtime is calculated in milliseconds using the `ctime` library.

Significance is also checked pairwise between the algorithms. The *t-values* and *p-values* are calculated subsequently while setting the significance level to 95%. Significance is verified if $p < 0.5$.

Monte Carlo Algorithm Results

MCA results were obtained as shown in Tables 6, 7 and 8. The results show average distance traveled, emission and runtime of each data file. As an example, the result obtained for the first data file is given in Tables 3, 4 and 5.

Iter	**1 - 10**	**11 - 20**	**21 - 30**	**31 - 40**	**41 - 50**	**51 - 60**	**61 - 70**	**71 - 80**	**81 - 90**	**91 - 100**
1	606641	550512	558721	502971	567591	521601	527721	563141	557802	571881
2	575431	561431	612981	603552	495071	574551	579161	592622	573951	519661
3	593992	579011	543531	598851	512961	624741	587241	542291	522581	554971
4	567551	559301	604681	529511	606341	561131	478521	542791	564511	594041
5	501741	548951	580251	599541	575761	596422	554961	545891	586021	523061
6	563871	601091	566561	549302	545591	576331	582101	577891	563721	595541
7	554031	546631	573161	595081	601141	567601	544881	479711	587112	527561
8	582351	501291	506021	550501	589741	571351	529301	621451	558781	564251
9	609521	617861	537521	567671	612921	592981	552221	578811	583111	588921
10	591241	553281	591311	543481	533091	522001	462421	536981	562231	575171

TABLE 3: Distance traveled in 100 iterations

Iter	1 - 10	11 - 20	21 - 30	31 - 40	41 - 50	51 - 60	61 - 70	71 - 80	81 - 90	91 - 100
1	30.86	28.01	28.42	25.59	28.88	26.54	26.85	28.65	28.38	29.09
2	29.27	28.56	31.18	30.7	25.19	29.23	29.46	30.15	29.2	26.44
3	30.22	29.46	27.65	30.47	26.1	31.78	29.88	27.59	26.58	28.23
4	28.87	28.45	30.76	26.94	30.85	28.55	24.34	27.61	28.72	30.22
5	25.52	27.93	29.52	30.5	29.29	30.34	28.23	27.77	29.81	26.61
6	28.69	30.58	28.82	27.94	27.76	29.32	29.61	29.4	28.68	30.3
7	28.19	27.81	29.16	30.27	30.58	28.88	27.72	24.4	29.87	26.84
8	29.63	25.5	25.74	28.01	30.01	29.07	26.93	31.62	28.43	28.71
9	31.01	31.43	27.35	28.88	31.18	30.17	28.09	29.45	29.66	29.99
10	30.08	28.15	30.08	27.65	27.12	26.56	23.52	27.32	28.6	29.26

TABLE 4: Emission in 100 iterations

Iter	1 - 10	11 - 20	21 - 30	31 - 40	41 - 50	51 - 60	61 - 70	71 - 80	81 - 90	91 - 100
1	2	1	15	2	15	11	2	1	2	2
2	2	2	2	19	3	2	11	12	5	8
3	8	17	18	2	2	2	2	10	4	5
4	10	2	13	3	10	15	5	2	6	8
5	3	2	2	14	2	2	2	15	14	1
6	13	14	4	1	2	6	14	2	5	2
7	5	1	2	2	17	1	2	2	8	13
8	2	10	6	18	5	16	1	14	4	1
9	13	16	2	2	7	1	18	3	2	3
10	2	3	14	2	8	2	2	16	15	17

TABLE 5: Time taken in msec for each iteration

City	Distance	Emission	Time	City	Distance	Emission	Time	City	Distance	Emission	Time
10	562848.48	28.64	6.61	**15**	1140121.45	58.01	3.85	**20**	1504160.8	76.53	3.93
10	735203.83	37.41	6.24	**15**	841036.55	42.79	3.05	**20**	1515113.01	77.09	3.97
10	609404.21	31.01	5.32	**15**	1203963.8	61.26	3.58	**20**	946586.41	48.16	4.17
10	688172.1	35.02	4.86	**15**	1053988.27	53.63	3.38	**20**	1283710.6	65.32	2.85
10	574340.14	29.22	4.65	**15**	1220929.4	62.12	3.18	**20**	1328122.8	67.58	3.55
10	725102.77	36.89	5.8	**15**	856318.44	43.57	2.82	**20**	1273106.4	64.78	4.12
10	681852.98	34.69	5.26	**15**	982703.86	50.1	3.5	**20**	847283.17	43.11	3.73
10	791488.76	40.27	5.94	**15**	674275.88	34.31	3.3	**20**	1249640.11	63.58	3.96
10	686631.5	34.94	5.91	**15**	910859.14	46.35	3.75	**20**	1525395.6	77.62	3.67
10	712311.38	36.24	3.73	**15**	826380.02	42.05	3.52	**20**	1351407.2	68.76	3.72
10	954043.71	48.54	4.81	**15**	1058919.22	53.88	3.8	**20**	1591005.1	80.95	3.37
10	550364.81	28.01	4.74	**15**	1242190.9	63.21	4.12	**20**	1394507.8	70.96	4.69
10	647078.12	32.92	4.45	**15**	974192.4	49.57	4.01	**20**	1409593.8	71.72	3.5
10	627570.8	31.93	5.97	**15**	1331462.3	67.75	3.63	**20**	1693841.5	86.19	4.22
10	400420.34	20.37	4.71	**15**	922543.01	46.94	3.21	**20**	1389642.9	70.71	3.48
10	521650.3	26.54	4.06	**15**	772136.21	39.29	2.98	**20**	1436985.1	73.12	3.49
10	653191.63	33.24	3.41	**15**	1121093.6	57.04	2.92	**20**	1274138.8	64.83	3.06
10	533967.38	27.17	3.66	**15**	1270343.4	64.64	3.09	**20**	1621922.2	82.53	3.38
10	641531.48	32.64	3.01	**15**	610765.01	31.08	3.54	**20**	1391410.7	70.8	4.11
10	489935.62	24.93	3.44	**15**	673422.13	34.27	4.19	**20**	1478435.9	75.23	4.16
Avg	639355.51	32.53	55.91		984382.24	50.08	69.42		1375300.46	69.97	75.13

TABLE 6: MCA Distance-Emission for cities 10, 15 and 20

Averages of the above data from Tables 3, 4 and 5 was *distance traveled: 562848.48*, *emission: 28.64* and *run-time: 6.61msec*, which is also the first line in the Table 6. The average values of each file is calculated in the same way. Furthermore, the final results for MCA consisting of standard deviation, average and total run-time is calculated as shown in Table 9 for distance traveled and Table 10 for emission.

City	Distance	Emission	Time	City	Distance	Emission	Time	City	Distance	Emission	Time
25	1414323.6	71.96	4.03	**50**	3352049.5	170.56	5.09	**75**	6365723.4	323.9	6.27
25	1447810.9	73.67	3.34	**50**	3644107.2	185.42	5.6	**75**	5348700.9	272.15	6.98
25	923696.54	47.1	3.65	**50**	3455875.1	175.84	5.63	**75**	5626950.9	286.31	6.66
25	1439744.9	73.26	3.95	**50**	3785465.9	192.61	5.51	**75**	5002886.7	254.56	6.49
25	1466229.7	74.6	3.75	**50**	3985782.7	202.80	4.84	**75**	6075322.9	309.12	6.42
25	1346242.1	68.5	4.31	**50**	3041769.5	154.77	6.66	**75**	5936251.1	302.05	6.52
25	1453560.8	73.96	3.24	**50**	3180509.2	161.83	5.89	**75**	5870727.4	298.71	6.83
25	1726658.1	87.86	4.02	**50**	3143719.3	159.96	6.13	**75**	6151205.7	312.99	6.86
25	1277454.3	65.1	4.27	**50**	4091421.01	208.18	5.28	**75**	6279125.7	319.49	6.49
25	1391840.1	70.82	4.27	**50**	3460430.01	176.07	4.59	**75**	6504391.8	330.96	6.3
25	1481923.6	75.4	4.24	**50**	3695201.4	188.02	5.93	**75**	4305113.6	219.05	6.67
25	1907110.7	97.04	3.55	**50**	3651248.3	185.78	6.04	**75**	6087695.1	309.75	6.34
25	921936.25	46.91	4.04	**50**	3273113.8	166.54	5.3	**75**	6821217.1	347.08	6.41
25	1802209.4	91.7	3.66	**50**	4269393.1	217.24	5.57	**75**	6172458.5	314.07	6.15
25	1875855.2	95.45	2.9	**50**	3654277.8	185.94	7.31	**75**	6739015.7	342.89	6.59
25	1546192.7	78.67	4.05	**50**	3345589.1	170.23	5.87	**75**	6588767.8	335.25	6.47
25	2202051.8	112.04	3.44	**50**	2229771.7	113.46	5.45	**75**	6011966.3	305.9	6.87
25	1895944.5	96.47	3.73	**50**	3757482.6	191.19	4.83	**75**	5570618.8	283.44	6.43
25	1934050.1	98.41	4.39	**50**	3252804.3	165.51	6.14	**75**	5731590.9	291.63	6.17
25	1509110.3	76.79	4.39	**50**	4151820.9	211.25	6.26	**75**	5920468.2	301.24	6.24
Avg	1548197.3	78.77	77.22		3521091.59	179.16	113.92		5955509.91	303.02	130.16

TABLE 7: MCA Distance-Emission for cities 25, 50 and 75

City	Distance	Emission	Time	City	Distance	Emission	Time	City	Distance	Emission	Time
100	8403267.7	427.57	10.47	**150**	10299704.3	524.07	15.23	**200**	14875182.12	756.88	20.98
100	8396677.6	427.24	7.79	**150**	12628627.7	642.57	14.23	**200**	15347354.34	780.9	20.94
100	6698961.4	340.86	8.73	**150**	9714527.1	494.29	14.48	**200**	14762177.45	751.13	21.42
100	7068508.6	359.66	8.62	**150**	11907099.1	605.86	14.27	**200**	13970840.12	710.86	22.16
100	6562330.6	333.9	7.85	**150**	9918542.6	504.67	15.35	**200**	16422443.3	835.61	22.42
100	8013215.1	407.73	9.55	**150**	10225822.2	520.31	14.67	**200**	13742943.4	699.27	21.88
100	8165304.2	415.47	8.5	**150**	991999.88	651.07	13.59	**200**	15020292.5	764.26	21.22
100	6716709.9	341.76	8.8	**150**	10054179.6	511.58	14.68	**200**	15383928.88	782.76	21.64
100	7908285.1	402.39	8.51	**150**	12189984.1	620.25	13.26	**200**	13722033.23	698.2	21.62
100	8296245.8	422.13	9.59	**150**	12610416.2	641.64	15.75	**200**	16785118.45	854.06	22.53
100	7020441.1	357.21	8.33	**150**	12472267.1	634.61	13.99	**200**	14286754.67	726.94	21.74
100	8537238.2	434.39	7.84	**150**	12535249.3	637.82	15.45	**200**	17078167.77	868.97	21.98
100	9416488.4	479.13	9.66	**150**	12499541.1	636.1	18.53	**200**	16717028.9	850.59	21.55
100	9260745.5	471.2	7.96	**150**	12436304.4	632.78	14.02	**200**	146184422.2	743.81	21.74
100	7160545.3	364.34	9.18	**150**	10236003.4	520.83	13.67	**200**	16195316.44	824.05	21.18
100	8325286.6	423.61	8.73	**150**	11628743.1	591.69	13.38	**200**	14918491.11	759.08	21.48
100	8484788.1	431.72	7.96	**150**	12157159.12	618.58	13.84	**200**	16959772.12	862.95	22.56
100	7062096.8	359.33	8.02	**150**	12432014.45	632.57	14.91	**200**	15054298.45	765.99	22.15
100	9153132.1	465.73	8.95	**150**	13668188.23	695.46	14.29	**200**	14234868.8	724.3	22.34
100	9465539.1	481.63	8.16	**150**	12936170.12	658.22	15.14	**200**	15936126.4	810.86	22.28
Avg	8005790.4	407.35	173.2		11177127.12	598.74	292.73		87662868.02	778.57	435.81

TABLE 8: MCA Distance-Emission for cities 100, 150 and 200

	Monte Carlo Algorithm		
Cities	STD	Average	Time
10	120115.79	639355.51	55.91
15	214595.02	984382.24	69.42
20	224450.92	1375300.46	75.13
25	325413.48	1548197.26	77.22
50	462335.74	3521091.59	113.92
75	597077.67	5955509.91	130.16
100	944899.33	8005790.36	173.2
150	2674406.02	11177127.12	292.73
200	323450532.7	87662868.02	425.81

TABLE 9: Distance traveled using MCA

	Monte Carlo Algorithm		
Cities	STD	Average	Time
10	6.11	32.53	55.91
15	10.41	50.08	69.42
20	10.92	69.97	75.13
25	16.55	78.77	77.22
50	22.92	179.16	113.92
75	30.38	303.02	130.16
100	48.07	407.35	173.2
150	61.56	598.74	292.73
200	75.57	778.57	425.81

TABLE 10: Emission using MCA

Discrete Differential Evolution Results

Results for DDE were not easy to obtain as the time taken to run the experiments was significantly longer than the other algorithms. The final results for DDE is shown in Table 12 and Table 13. These tables are also the extension for MCA Tables 9 and 10, which was added to check if the distance traveled, emission and run time improved or not. Table 11, shows the averages of distance traveled, emission and time of each file of 10 cities comparing only the MCA and DDE algorithms.

	Distance		**Emission**		**Runtime**	
Data File	**MCA**	**DDE**	**MCA**	**DDE**	**MCA**	**DDE**
10_1	562848.48	**420569.3**	28.64	**21.39**	**5.94**	14613.23
10_2	735203.83	**532089.4**	37.41	**27.07**	**6.24**	8181.7
10_3	609404.21	**426113.2**	31.01	**21.68**	**5.32**	8096.63
10_4	688172.1	**500723.7**	35.02	**25.47**	**4.86**	15293.4
10_5	574340.14	**505101.4**	29.22	**25.70**	**4.65**	9700.72
10_6	725102.77	**469960.2**	36.89	**23.91**	**5.8**	8796.14
10_7	681852.98	**505101.4**	34.69	**25.87**	**5.26**	13884.09
10_8	791488.76	**531127.8**	40.27	**27.02**	**5.94**	14214.1
10_9	686631.5	**506836.3**	34.94	**25.78**	**5.91**	7899.77
10_10	712311.38	**488064.4**	36.24	**24.83**	**3.73**	14600.22
10_11	954043.71	**699959.5**	48.54	**35.61**	**4.81**	14567.22
10_12	550364.81	**326091.1**	28.01	**16.59**	**4.74**	14573.15
10_13	647078.12	**427385.2**	32.92	**21.74**	**4.45**	11581.96
10_14	627570.8	**433822.1**	31.93	**22.07**	**5.97**	18073.74
10_15	400420.34	**260930.6**	20.37	**13.27**	**4.71**	13828.31
10_16	521650.3	**407672.5**	26.54	**20.74**	**4.06**	8916.12
10_17	653191.63	**448224.2**	33.24	**22.80**	**3.41**	9338.92
10_18	533967.38	**401943.7**	27.17	**20.45**	**3.66**	9605.8
10_19	641531.48	**476465.1**	32.64	**24.24**	**3.01**	9637.28
10_20	489935.62	**295896.5**	24.93	**15.05**	**3.44**	12273.13

TABLE 11: DDE: Distance traveled, Emission, runtime for each file of 10 cities.

	Monte Carlo Algorithm			Discrete Differential Evolution			Significance		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	120115.79	639355.51	**55.91**	**94918.36**	**453203.8**	237675	4.29	0.00014	Y
15	214595.02	984382.24	**69.42**	**13172.89**	**526779.24**	257181	4.07	0.00034	Y
20	224450.92	1375300.46	**75.13**	**147022.12**	**755195.6**	300708	5.37	0.00001	Y
25	325413.48	1548197.26	**77.22**	**20391.93**	**1041373.34**	342999	3.43	0.00036	Y
50	462335.74	3521091.59	**113.92**	**444452.64**	**2694902.7**	454780	1.96	0.02528	Y
75	597077.67	5955509.91	**130.16**	**577120.56**	**4126600.9**	503447	2.45	0.0074	Y
100	944899.33	8005790.36	**173.2**	**620822.28**	**5778088.2**	548122	7.61	0.00001	Y
150	2674406.02	11177127.12	**292.73**	**969757.57**	**8048146.5**	708222	8.46	0.00001	Y
200	323450532.7	87662868.02	**425.81**	**1032876.54**	**12910399.1**	1007201	9.16	0.00001	Y
Avg	36557091.84	13429958.05	**157.05**	**435614.98**	**4037187.7**	484481.56			

TABLE 12: DDE: Distance traveled

	Monte Carlo Algorithm			Discrete Differential Evolution			Significance		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	6.11	32.53	**55.91**	**4.25**	**19.91**	237675	4.29	0.00014	Y
15	10.41	50.08	**69.42**	**4.36**	**33.65**	257181	4.07	0.00034	Y
20	10.92	69.97	**75.13**	**6.64**	**45.88**	300708	5.37	0.00001	Y
25	16.55	78.77	**77.22**	**10.33**	**50.42**	342999	3.43	0.00036	Y
50	22.92	179.16	**113.92**	**24.48**	**136.03**	454780	1.96	0.02528	Y
75	30.38	303.02	**130.16**	**30.12**	**226.3**	503447	2.45	0.0074	Y
100	48.07	407.35	**173.2**	**39.67**	**327.67**	548122	7.61	0.00001	Y
150	61.56	598.74	**292.73**	**46.73**	**504.84**	708222	8.46	0.00001	Y
200	75.57	778.57	**425.81**	**55.61**	**678.01**	1007201	9.16	0.00001	Y
Avg	31.38	277.57	**157.05**	**24.68**	**224.74**	484481.56			

TABLE 13: DDE: Emission

Discrete Particle Swarm Optimization Results

DPSO was similar to DDE as it is another optimization technique used to solve the PRP problem. The results for DPSO can be see in Tables 15 and 16. The time taken shows a significant improvement when compared to DDE.

	Distance		Emission		Runtime	
Data File	**MCA**	**DPSO**	**MCA**	**DPSO**	**MCA**	**DPSO**
10_1	562848.5	**440836.6**	28.64	**22.43**	**5.94**	30.05
10_2	735203.8	**556816.4**	37.41	**28.33**	**6.24**	25.17
10_3	609404.2	**479242.7**	31.01	**24.38**	**5.32**	28.42
10_4	688172.1	**528182.01**	35.02	**26.87**	**4.86**	31.02
10_5	574340.1	**506645.34**	29.22	**25.78**	**4.65**	24.98
10_6	725102.8	**566794.8**	36.89	**28.84**	**5.8**	27.74
10_7	681853.33	**525973.1**	34.69	**26.76**	**5.26**	26.49
10_8	791488.8	**650787.5**	40.27	**33.11**	**5.94**	27.87
10_9	686631.5	**518828.5**	34.94	**26.4**	**5.91**	26.36
10_10	712311.4	**517874.6**	36.24	**26.35**	**3.73**	23.61
10_11	954043.7	**703570.3**	48.54	**35.8**	**4.81**	25.41
10_12	550364.8	**415718.9**	28.01	**21.15**	**4.74**	27.59
10_13	647078.1	**526970.9**	32.92	**26.81**	**4.45**	30.1
10_14	627570.8	**497145.3**	31.93	**25.3**	**5.97**	27.07
10_15	400420.3	**301968.1**	20.37	**15.36**	**4.71**	22.25
10_16	521650.3	**447942.7**	26.54	**22.79**	**4.06**	22.33
10_17	653191.6	**505155.9**	33.24	**25.7**	**3.41**	21.79
10_18	533967.4	**407903.6**	27.17	**20.75**	**3.66**	20.29
10_19	641531.5	**476782.4**	32.64	**24.26**	**3.01**	21.03
10_20	489935.6	**368398.2**	24.93	**18.74**	**3.44**	21.94

TABLE 14: DPSO: Distance traveled, Emission, runtime for each file of 10 cities.

	Monte Carlo Algorithm			Discrete Particle Swarm			Significance		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	120115.79	639355.51	**55.91**	**90016.96**	**497176.87**	411.51	5.64	0.00001	Y
15	214595.02	984382.24	**69.42**	**155883.95**	**701477.56**	467.57	4.07	0.00024	Y
20	224450.92	1375300.46	**75.13**	**157691.37**	**938290.34**	491.23	5.37	0.00001	Y
25	325413.48	1548197.26	**77.22**	**242939.65**	**1065200.45**	551.41	7.46	0.00561	Y
50	462335.74	3521091.59	**113.92**	**355254.78**	**2598102.52**	862.44	4.96	0.04284	Y
75	597077.67	5955509.91	**130.16**	**551396.35**	**4703112.82**	1230.73	2.43	0.04401	Y
100	944899.33	8005790.36	**173.2**	**732605.86**	**6553508.39**	1459.08	7.61	0.00001	Y
150	2674406.02	11177127.12	**292.73**	**977478.52**	**10023414.6**	2253.74	8.57	0.00981	Y
200	323450532.7	87662868.02	**425.81**	**1047712.34**	**13507528.6**	3240.55	9.16	0.00001	Y
Avg	36557091.84	13429958.05	**157.05**	**478997.75**	**4509756.9**	1218.69			

TABLE 15: DPSO: Distance traveled

	Monte Carlo Algorithm			Discrete Particle Swarm			Significance		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	6.11	32.53	**55.91**	**4.58**	**25.29**	411.51	5.64	0.00001	Y
15	10.41	50.08	**69.42**	**7.93**	**35.69**	467.57	4.07	0.00024	Y
20	10.92	69.97	**75.13**	**8.02**	**47.74**	491.23	5.37	0.00001	Y
25	16.55	78.77	**77.22**	**12.36**	**54.2**	551.41	7.46	0.00561	Y
50	22.92	179.16	**113.92**	**18.07**	**132.19**	862.44	4.96	0.04284	Y
75	30.38	303.02	**130.16**	**28.05**	**239.3**	1230.73	2.43	0.04401	Y
100	48.07	407.35	**173.2**	**37.27**	**333.45**	1459.08	7.61	0.00001	Y
150	61.56	598.74	**292.73**	**49.73**	**510.01**	2253.74	8.57	0.00981	Y
200	75.57	778.57	**425.81**	**57.56**	**687.28**	3240.55	9.16	0.00001	Y
Avg	31.38	277.57	**157.05**	**24.84**	**229.46**	1218.69			

TABLE 16: DPSO: Emission

Differential Evolution Algorithm and Discrete Particle Swarm Algorithm Result Comparison

While comparing the MCA with two widely used optimized techniques, the objective was to also ascertain as to which of these algorithms is better performing. Table 17 shows the distance traveled comparison between DDE and DPSO. Table 18, shows the emission comparison between DDE and DPSO.

	DDE			**DPSO**			**Significance**		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	**94918.36**	**453203.8**	237675	90016.96	497176.87	**411.51**	-111.61	0.00001	Y
15	**13172.89**	**526779.24**	257181	155883.95	701477.56	**467.57**	27.28	0.00001	Y
20	**147022.12**	**755195.6**	300708	157691.37	938290.34	**491.23**	38.05	0.00001	Y
25	**20391.93**	**1041373.34**	342999	242939.65	1065200.45	**551.41**	-6.93	0.00001	Y
50	444452.64	2694902.7	454780	**355254.78**	**2598102.52**	**862.44**	-14.25	0.00001	Y
75	577120.56	**4126600.9**	503447	**551396.35**	4703112.82	**1230.73**	24.59	0.00001	Y
100	**620822.28**	**5778088.2**	548122	732605.86	6553508.39	**1459.08**	28.77	0.00001	Y
150	**969757.57**	**8048146.5**	708222	977478.52	10023414.6	**2253.74**	6.68	0.00001	Y
200	**1032876.54**	**12910399.1**	1007201	1047712.34	13507528.6	**3240.55**	2.43	0.00001	Y
Avg	**435614.98**	**4037187.7**	484481.6	478997.75	4509756.9	**1218.69**			

TABLE 17: Distance traveled comparison between DDE and DPSO

	DDE			**DPSO**			**Significance**		
Cities	**STD**	**Average**	**Time**	**STD**	**Average**	**Time**	***t***	***p***	***p* <0.05**
10	**4.25**	**19.91**	237675	4.58	25.29	**411.51**	-111.61	0.00001	Y
15	**4.36**	**33.65**	257181	7.93	35.69	**467.57**	27.28	0.00001	Y
20	**6.64**	**45.88**	300708	8.02	47.74	**491.23**	38.05	0.00001	Y
25	**10.33**	**50.42**	342999	12.36	54.2	**551.41**	-6.93	0.00001	Y
50	**24.48**	136.03	454780	18.07	**132.19**	**862.44**	-14.25	0.00001	Y
75	**30.12**	**226.3**	503447	28.05	239.3	**1230.73**	24.59	0.00001	Y
100	39.67	**327.67**	548122	**37.27**	333.45	**1459.08**	28.77	0.00001	Y
150	**46.73**	**504.84**	708222	49.73	510.01	**2253.74**	6.68	0.00001	Y
200	**55.61**	**678.01**	1007201	57.56	687.28	**3240.55**	2.43	0.00001	Y
Avg	**24.68**	**224.74**	484481.6	24.84	229.46	**1218.69**			

TABLE 18: Emission comparison between DDE and DPSO

CHAPTER VI

ANALYSIS

This section analyses the results obtained using MCA, DDE and DPSO in the previous section. The results of DDE and DPSO show significant improvement over MCA. As an example, Fig 14, Fig 15 and Fig 16 are generated using the Uk_01 dataset to show the progression of the algorithm over a number of iterations. These figures are generated using the `tkinter` Python code. The parameters for the results obtained for Fig 14, Fig 15 and Fig 16 are:

- Number of rows: 10
- Number of iterations: 30
- Number of columns : 10
- Algorithm selected: DDE, DPSO and MCA

Distance Traveled Analysis

The first objective function to be analyzed is the distance traveled, and the results are given in Tables 11 and 12 for DDE algorithm and Tables 14 and 15 for the DPSO algorithms. These four tables compare the results against the MCA method.

Both DDE and DPSO results significantly improve on the MCA results. This is due to the fact that evolutionary algorithms are based on directed search and not random behavior such as the MCA. DDE improved on all results for all data instances.

In Table 14, the first data set of ten cities is analyzed. In all the results, the DDE algorithm is better, however it requires extra runtime. Table 12 compares all the datasets

grouped in the city sizes. The average (4037187.7) and standard deviation (435614.98) for DDE is shown to be better compared to the MCA algorithms average (13429958.05) and standard deviation (36557091.84) value. The execution time however is better for the MCA algorithm of 157.05*msec* compared to 484481.56*msec* for the DDE algorithm.

Significance test at 95% confidence interval is done between the two average values, and for all of the instances, the DDE algorithm is shown to be significantly better than the MCA algorithm.

The first DPSO distance results is given in Table 14 for the first ten cities. As in the previous case of the DDE algorithm, the DPSO algorithm performed better than the MCA algorithm, however with a negative influence of runtime. The DPSO distance results for all datasets is given in Table 15. For all the problems instances, DPSO obtains a better standard deviation (478997.75) and average (4509756.9) values compared to MCA standard deviation (36557091.84) and average (13429958.05) values. The overall average time is however higher for DPSO (1218.69*msec* compared to MCA runtime (157.05*msec*). The final pairwise comparison is done between the DDE algorithm and the DPSO algorithm as given in Table 17. From the results obtained, the DDE algorithm is better performing in all the datasets apart from the 50 cities. The average standard deviation for DDE is 435614.98 compared to 478997.75 for DPSO, which is a 9% improvement. The average value of DDE is 4037187.7 compared to 4509756.9 for DPSO, which is a 10.47% improvement of DDE over DPSO. The big drawback of DDE is its runtime which is significantly longer than both MCA and DPSO algorithms.

An example of the iterations of the three algorithms is given in Fig 14 on a sample 10 city problem. The improvement of average values is shown over the many iterations to showcase how the algorithms operate.

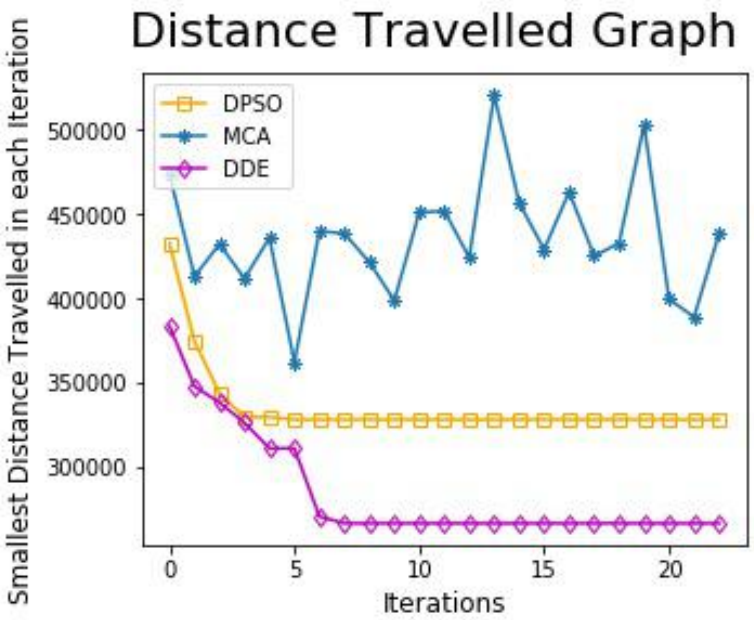

FIGURE 14: Distance graph for 10 cities

Emission Analysis

The second objective function to be analyzed is the emission, and the results are given in Tables 11 and 13 for DDE algorithm and Tables 14 and 16 for the DPSO algorithms. These four tables compare the emission results against the MCA method.

As in the previous case, both DDE and DPSO results significantly improve on the MCA emission results.

In Table 14, the first data set of ten cities is analyzed. In all the results, the DDE algorithm obtains better emission results, however it requires extra runtime. Table 13 compares all the datasets grouped in the city sizes. The average (224.74) and standard deviation (24.68) for DDE is shown to be better compared to the MCA algorithms average (277.57) and standard deviation (31.38) value. The execution time however is better for the MCA algorithm of 157.05*msec* compared to 484481.56*msec* for the DDE algorithm.

Significance test at 95% confidence interval is done between the two average emission values, and for all of the instances, the DDE algorithm is shown to be significantly better than the MCA algorithm.

The first DPSO emission results is given in Table 14 for the first ten cities. The DPSO algorithm performed better on all data instances than the MCA algorithm.
The DPSO emission results for all datasets is given in Table 16. For all the problems instances, DPSO obtains a better standard deviation (24.84) and average (229.46) values compared to MCA standard deviation (31.38) and average (277.57) values.

The final pairwise comparison is done between the DDE algorithm and the DPSO algorithm for emission is given in Table 18. From the results obtained, the DDE algorithm is better performing in all the datasets apart from the 50 cities. The average standard deviation for DDE is 24.68 compared to 24.84 for DPSO, which is a 0.64% improvement. The average value of DDE is 224.74 compared to 229.46 for DPSO, which is a 0.02% improvement of DDE over DPSO. All significance test at 95% confidence interval shows that the algorithm obtaining a better value significantly improves on the other algorithm for that dataset. Overall, it can be stated that DDE is a better performing algorithm.

An example of the iterations of the three algorithms is given in Fig 15, on a sample 10 city problem. The improvement of average emission values is shown over the many iterations to showcase how the algorithms operate.

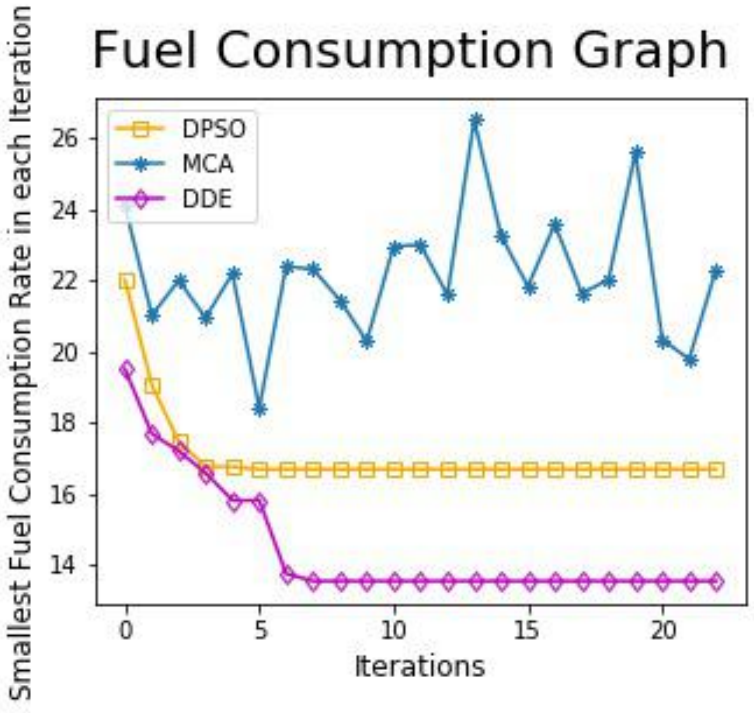

FIGURE 15: Emission graph for 10 cities

Algorithm Runtime Analysis

The final analysis is of the algorithm runtimes. The MCA algorithm takes on average **157.05***msec* to run, whereas DDE takes **484481.56***msec* and DPSO takes **1218.69***msec* to run the datasets. We can conclude that MCA and DPSO are relatively comparable, whereas DDE is significantly longer in runtime. This can be easily explained with the fact that DDE has a NEH local search routine, whereas MCA and DPSO do not employ any local search routines. This is a tradeoff between the different algorithms, and also is influenced by the machine architecture that is being used to run the experimentations.

An example is given in Fig 16, where the time taken by DPSO was much better than DDE. The runtime of DPSO is almost the same as for MCA.

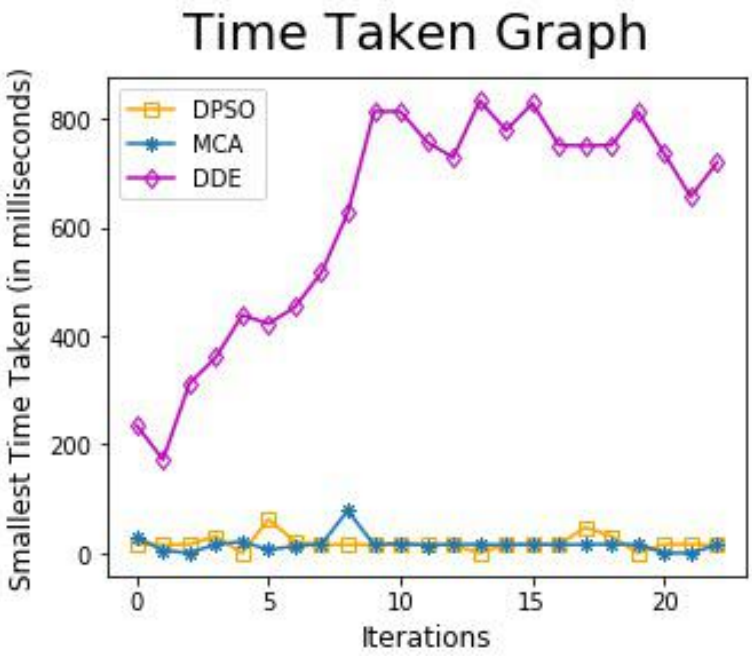

FIGURE 16: Algorithm runtime graph for 10 cities

CHAPTER VII

CONCLUSION

The aim of this project was to solve a problem that has been identified as one of the biggest challenges that humanity is facing, that of reducing carbon emissions [21]. Air pollution is been seen as a major factor in a lot of the societal problems in the world. These problems include reduced population density, species density, and species richness in communities [22]. Some of these problems have no solution (or none yet) and for some, the extent of the problems is still yet to be determined. One of the key components that has been identified as a major contributor for air pollution is vehicular traffic, specifically commercial vehicles.

The motivation of this project was to find suitable algorithms that can be used to solve Vehicle Routing Problems (VRP), specifically the pollution focused Pollution Routing Problem (PRP). There are many papers written on the PRP, however each is taking a fixed approach which cannot be adapted to other modification of the problem.

The idea was then to use advanced evolutionary algorithms and test its effectiveness to solve this problem. Initially, the project started by using the traditional way of calculating the distance traveled formula and adding emission formula (FCR) on it.

Therefore, VRP was transformed into the PRP problem and a suitable dataset from the UK was utilized for testing purposes. Initially, by using MCA algorithm, major improvements were seen in the results over the iterations. For the data file Uk10_1, the distance traveled from the classic sequence of city was *847741* and emission was *43.1437*, whereas using MCA gives distance traveled as *559193* and emission as *28.45*. Using the MCA algorithm, the emission value were improved by almost 1.5 times.

Applying the two different evolutionary algorithm of DDE and DPSO further improved on these results. Both these algorithms were adapted from literature, where they had been used to solve strict permutative based combinatorial optimization problems. These algorithms were modified to solve the PRP problem.

Two separate analysis were done for distance traveled and emission. Aside from one dataset, DDE was the best performing algorithm overall with the DPSO ahead of the MCA algorithm. By employing a local search routine inside DPSO, it is believed that the performance will be on par with DDE.

The application of these evolutionary algorithms validate the aim of the project that different algorithms can effectively solve the PRP problem and they can be easily adapted and scaled to solve such problems from other regions and states across the globe. The reduction in pollutants remains a key objective for humanity.

Future Expectations

- The future work can include using more optimizing algorithms to see if better results can be obtained.
- High performance paradigms such as *CUDA* can be used to accelerate these algorithms for better in-time analysis of conditions.
- A model can be created which for example uses the PRP as a layer for Google Maps to indicate which route will cause less pollution for traffic analysis and route setting purposes.
- Instead of using `tkinter` forms, an app can be created. Trucking companies can use it for optimizing the route and improving the environment thereby creating less pollution and improving air quality.

REFERENCES CITED

[1] T. Bektaş and G. Laporte, "The pollution-routing problem," *Transportation Research Part B: Methodological*, vol. 45, no. 8, pp. 1232–1250, 2011.

[2] R. Kramer, A. Subramanian, T. Vidal, and F. C. Lucídio dos Anjos, "A matheuristic approach for the pollution-routing problem," *European Journal of Operational Research*, vol. 243, no. 2, pp. 523–539, 2015.

[3] M. Figliozzi, "Vehicle routing problem for emissions minimization," *Transportation Research Record*, vol. 2197, no. 1, pp. 1–7, 2010.

[4] A. McKinnon, "co_2 emissions from freight transport in the uk," *Commission for Integrated Transport, London*, 2007.

[5] E. Demir, T. Bektaş, and G. Laporte, "A comparative analysis of several vehicle emission models for road freight transportation," *Transportation Research Part D: Transport and Environment*, vol. 16, no. 5, pp. 347–357, 2011.

[6] H. R. Kirby, B. Hutton, R. W. McQuaid, R. Raeside, and X. Zhang, "Modelling the effects of transport policy levers on fuel efficiency and national fuel consumption," *Transportation Research Part D: Transport and Environment*, vol. 5, no. 4, pp. 265–282, 2000.

[7] N. Tajik, R. Tavakkoli-Moghaddam, B. Vahdani, and S. M. Mousavi, "A robust optimization approach for pollution routing problem with pickup and delivery under uncertainty," *Journal of Manufacturing Systems*, vol. 33, no. 2, pp. 277–286, 2014.

[8] R. G. Derwent and O. Hov, "Computer modeling studies of the impact of vehicle exhaust emission controls on photochemical air pollution formation in the united kingdom," *Environmental Science & Technology*, vol. 14, no. 11, pp. 1360–1366, 1980.

[9] R. S. Kumar, K. Kondapaneni, V. Dixit, A. Goswami, L. S. Thakur, and M. Tiwari, "Multi-objective modeling of production and pollution routing problem with time window: A self-learning particle swarm optimization approach," *Computers & Industrial Engineering*, vol. 99, pp. 29–40, 2016.

[10] S. Goyal, S. Ghatge, P. Nema, and S. Tamhane, "Understanding urban vehicular pollution problem vis-a-vis ambient air quality–case study of a megacity (delhi, india)," *Environmental monitoring and assessment*, vol. 119, no. 1-3, pp. 557–569, 2006.

[11] D. Davendra, I. Zelinka, M. Bialic-Davendra, R. Senkerik, and R. Jasek, "Discrete self organising migrating algorithm for the task of capacitated vehicle routing problem," *Mendel*, pp. 259–265, 01 2011.

[12] R. Nath, A. Rauniyar, P. K. Muhuri, and A. K. Shukla, "A novel bilevel formulation for pollution routing problem," pp. 586–562, 2018.

[13] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete differential evolution algorithm for the permutation flowshop scheduling problem," *Computers & Industrial Engineering*, vol. 55, no. 4, pp. 795–816, 2008.

[14] X. Wang and L. Tang, "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking," *Applied Soft Computing*, vol. 12, no. 2, pp. 652–662, 2012.

[15] U. of Southampton, "The pollution-routing problem instance library." `http://www.apollo.management.soton.ac.uk/prplib.htm`, 2020.

[16] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, p. 3–30, Jan. 1998.

[17] V. Sriram and D. Kearney, "An area time efficient field programmable mersenne twister uniform random number generator," in *ERSA*, pp. 244–246, Citeseer, 2006.

[18] M. Nawaz, E. E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[19] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *European journal of operational research*, vol. 177, no. 3, pp. 1930–1947, 2007.

[20] J. W. Shipman, "Tkinter 8.4 reference: a gui for python," *New Mexico Tech Computer Center*, 2013.

[21] J. Kagawa, "Atmospheric pollution due to mobile sources and effects on human health in japan.," *Environmental health perspectives*, vol. 102, no. suppl 4, pp. 93–99, 1994.

[22] P. Movalli, O. Krone, D. Osborn, and D. Pain, "Monitoring contaminants, emerging infectious diseases and environmental change with raptors, and links to human health," *Bird Study*, vol. 65, no. sup1, pp. S96–S109, 2018.